E190AR: Lab 3 Report

Charles Dawson & Peter Johnson

***Abstract*— This report describes the implementation and validation of a feedback-control based point-tracking and path-following functionality for our E190AR robot. This functionality allows our robot to consistently track points as far as 1 meter away with centimeter-level accuracy, laying the foundation for future work implementing path planning and autonomous navigation. The theoretical basis for our feedback control system is also discussed.**

I. Introduction

The overall goal of an autonomous robot navigation system is to be able to simply provide the robot with a target location and let the robot determine how to move in order to achieve that goal. This requires not only high-level global path planning and low-level motion control, but also intermediate, or local, path planning. Local path planning is the mechanism by which the robot navigates between adjacent waypoints along a longer route determined by a global path planning algorithm (developed in a future lab).

An example of this path planning task is shown in Fig. 1, where the global path planning algorithm determines the path from the starting position to the goal (shown as the black line) while the local path planning tracks the robot towards intermediate waypoints (shown in blue). The local point tracking process is ignorant of obstacles along the path, as it assumes that the global planning process has generated a sequence of waypoints that avoids any obstacles), and the local point tracking also assumes that the waypoints are sufficiently close together that the robot control system can be treated using appropriate linearizations.

II. Theory

This section presents the mathematical theory behind a feeback control system for local robot point tracking.

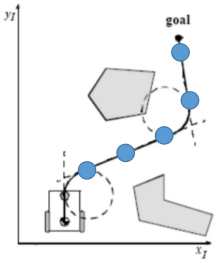

Fig. 1. Example of the interaction between global and local path planning.

Fig. 2 shows the definitions used for this control task. Given a goal position (X_G, Y_G) in the global frame (which we take to be the origin of the global frame without loss of generality for the analysis of this control task, following Seigwart et al [2]), the robot position $(\Delta X, \Delta Y)$ and orientation θ, we define the error in robot position $e = [\Delta X, \Delta Y, \theta]^T$, which we seek to minimize. To do this, we introduce a convenient choice of state vector as in [2] and [1]: $\mathbf{x} = [\rho, \alpha, \beta]^T$, where

$$\rho = \sqrt{\Delta X^2 + \Delta Y^2} \tag{1}$$

$$= -\theta + \text{atan2}(\Delta Y, \Delta X) \tag{2}$$

$$\beta = -\theta - \alpha \tag{3}$$

Based on the kinematics of the differential drive robot, we know that $\mathbf{x}$ is governed by the dynamic system

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -\cos\alpha & 0 \\ \sin(\alpha)/\rho & -1 \\ -\sin(\alpha)/rho & 0 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix}, \tag{4}$$

where v and w are the control inputs (linear velocity in the x direction and angular velocity in the z direction, respectively) and the angle α is assumed to fall between $-\pi/2$ and $\pi/2$.

We define a simple proportional control law $v = k_\rho \rho$ and $w = k_\alpha \alpha + k_\beta \beta$, which yields full system dynamics

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -k_\rho \rho \cos\alpha \\ k_\rho \sin(\alpha) - k_\alpha \alpha - k_\beta \beta \\ -k_\rho \sin(\alpha) \end{bmatrix}, \tag{5}$$

which we can linearize about $\alpha = 0$ to get the linear system

$$\begin{bmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{bmatrix} = \begin{bmatrix} -k_\rho & 0 & 0 \\ 0 & -(k_\alpha - k_\rho) & -k_\beta \\ 0 & -k_\rho & 0 \end{bmatrix} \begin{bmatrix} \rho \\ \alpha \\ \beta \end{bmatrix}. \tag{6}$$

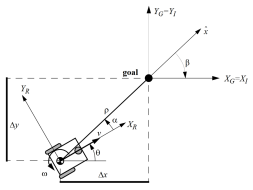

Fig. 2. Definition of local point tracking control task.

The eigenvalues of the gain matrix in this system are given by the roots of the characteristic polynomial

$$(\lambda + k_\rho)(\lambda^2 + \lambda(k_\alpha - k_\rho) - k_\rho k_\beta) = 0 \tag{7}$$

For this system to be stable, all the roots of this equation must have real parts less than zero, which yields the stability criteria:

$$k_\rho > 0; k_\beta < 0; k_\alpha - k_\rho > 0 \tag{8}$$

Additionally, the lecture slides [1] also provide the "strong stability condition":

$$k_\rho > 0; k_\alpha < 0; k_\alpha + \frac{5}{3}k_\beta - \frac{2}{\pi}k_\rho > 0, \tag{9}$$

which we used to select control gains for our robot, as discussed section IV.

III. Point Tracking Node Implementation

This section describes the implementation of the point tracking node. The `/cmd_vel` topic is subscribed to by our robot control node (discussed in previous lab reports) and is how our robot receives the control inputs v and w determined by the feedback control laws discussed above. The `/cmd_vel` messages are produced and published by our point tracking node. The point tracking node calculates the `/cmd_vel` based on a target point of type `geometry_msgs/Pose` using the feedback control laws described in section II. The Pose consists of a Point position in the global frame (`odom_wheel`) and a Quaternion orientation of the robot at that point. The node takes in these target points by subscribing to `/goal_path` which is of type `nav_msgs/Path`. Paths are composed of `geometry_msgs/PoseStamped[]`, an array of stamped poses with the first pose to be targeted in index 0 and so on. For an example Path message, see Fig. 3. In this case, the robot would go forwards to (1,0) and then left to (1,1) (both in the global frame).

The point tracking code is written so the stopping threshold for the robots calculated distance from its target point is 0.05 m (if the robot odometry calculates that the robot is within 5 cm of its target in the global frame, it will stop because it is close enough.) At this point, the robot can then move on to the next target in its path. This was implemented to prevent the robot from wasting time trying to get closer to a point it might not be able to reach. The loss in precision due to this cutoff was deemed acceptable, due to the necessity of moving onto the next target promptly. Once this threshold has been reached, the point tracking node updates the index of the target point, and the target point is then updated to match that of the current index in the path. When the target was updated, we also needed to invalidate the previous ρ value (distance to the target) to be -1, because the previously calculated value would lead to erroneous control inputs for the newly selected target. This allowed us to add a state to our `cmd_vel_pub` function to set our

```
rostopic pub /goal_Path nav_msgs/Path "header:
  seq: 0
  stamp:
    secs: 0
    nsecs: 0
  frame_id: '/odom_wheel'
poses:
- header:
    seq: 0
    stamp:
      secs: 0
      nsecs: 0
    frame_id: '/odom_wheel'
  pose:
    position:
      x: 1.0
      y: 0.0
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 0.0
- header:
    seq: 0
    stamp:
      secs: 0
      nsecs: 0
    frame_id: '/odom_wheel'
  pose:
    position:
      x: 1.0
      y: 1.0
      z: 0.0
    orientation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 0.0"
```

Fig. 3. Command used to instruct the robot to move forward and then left to (1,1).

x velocity and z angular velocity to 0 so the robot would not move while the target point was being updated. Pseudocode showing the flow of program execution in given in Listing. 1.

Listing 1. Point Tracking Pseudocode

```
for point in path:
    - Update rho, alpha, beta when
        odometry is retrieved
    Every 5 Hz:
        - Update v, w based on control
            law and previously calculated
             rho, alpha, beta
        if rho < 0:
            # This means rho has been
                invalidated by a previous
                 update, so do nothing
            v, w = 0
        elif rho < 0.05
            # This means we're close
                enough to the target to
                move on to the next point
            v, w = 0
            current_path_index = min(
                current_path_index, len(
                path) - 1)
            if path is not empty:
                rho = -1 # invalidate
                    rho
```

```
            current_target = path[
                current_path_index]
    - Publish \cmd_vel with v and w
```

Additionally, we modified the control.py code per Giselle Serate and David Sobek's recommendation to add a timeout to the XBee receive function, preventing control.py from crashing whenever the XBee radio dropped a packet.

IV. Experimental Calibration

This section describes the procedures used to calibrate the k_ρ, k_α, and k_β gain constants for our robot, as well as the testing done to determine the accuracy and precision of our point tracking along a path. We calibrated the control gain constants by picking initial values within the bounds of the strong stability criteria given in section II, then refining them as needed to improve the performance. The biggest modification we needed involved reducing the value of k_ρ relative to k_α and k_β, because the control input v has units of m/s, which is qualitatively much faster per unit than w, which has units of rad/s. However, we found that we could not reduce k_ρ too far because we risked entering the deadband of the robot's motors. In the end, we selected gain constants $k_\rho = 0.5$, $k_\beta = 0.25$, and $k_\alpha = -0.75$. We also placed limits on both the maximum and minimum PWM values that were being sent to the robot.

Using these gain constants, we tested the performance of our local point tracking node by commanding the robot to move along the path from (0, 0) to (1, 0) to (1, 1) in the global frame. This path has a total length of 2 m, allowing us to calculate the accuracy and precision of the point tracking function relative to the 2 m distance travelled. The results from this test are shown graphically in Fig. 4 and tabulated in Table 5, which shows the error between the final robot position and the final goal position (1, 1) as $(\delta x, \delta y)$, along with the percent error associated with our point tracking ($100\% \frac{\rho}{2\ m}$. On average, our robot tracked the target position to within 11.4 cm (5.74%), meaning that the compound error in our odometry over the 2 m path was less than 7 cm (considering the 5 cm stopping threshold built in to our point tracking node). The standard deviation in distance to the stopping point was 3.9 cm, suggesting that our robot (which itself has a 7 cm chassis radius) is fairly consistent (as well as reasonably accurate) in accomplishing its local point tracking task.

V. Conclusions and Reflection

Based on the work presented above, our robot is now able to

1) Navigate to the vicinity of a point in the global coordinate system in an arbitrary linear combinations of translational x motion and rotational z motion
2) Follow a path consisting of multiple points in the global coordinate system, received by subscribing to a ROS topic

These capabilities lay the foundation for future work implementing path planning, obstacle detection, and autonomous navigation which will be needed for the final project.

References

[1] Y. Chang. E190AR. Class Lecture, Topic: "Point Tracking." Department of Engineering, Harvey Mudd College, Claremont, CA. Slides from C. Clark.

[2] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. Cambridge, MA: MIT, 2011.

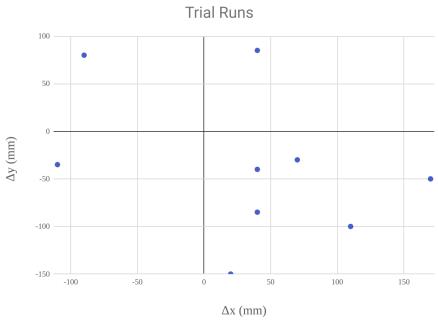

Fig. 4. Distance of robot from target for each trial run

Trial #	Δx (mm)	Δy (mm)	% Error in ρ
1	40	-40	2.83%
2	70	-30	3.81%
3	40	-85	4.70%
4	20	-150	7.57%
5	170	-50	8.86%
6	40	85	4.70%
7	110	-100	7.43%
8	-90	80	6.02%
9	-110	-35	5.77%
Avg	32.2	-36.1	5.74%

Fig. 5. Tests of point tracking and path following. All distances in mm.